LIST

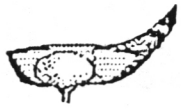
LONG ISLAND SINCLAIR TIMEX GROUP
INCORPORATING * NYTSE OF NEW YORK CITY
ISSUE: SUMMER 1990

* NEW YORK TIMEX SINCLAIR ENTHUSIASTS: NEXT MEETING SEPT SEP 16 1990
SEP. 16 1990

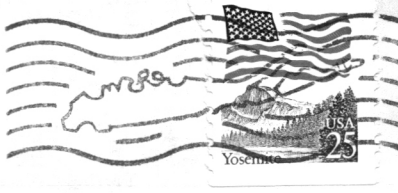
DISK DRIVES SUPPORTED:
1 OLIGER
2 LARKEN
3 AERCO

LIST MEMBERSHIP IS $15.00. LIBRARY TAPES ARE AVALIABLE, WRITE THE ADDRESS PRINTED BELOW.

BASIC AT A STRETCH



L.I.S.T.
5 PERI LANE
VALLEY STREAM, NY
11581



TO: *Don Lambert JAN/91*
3310 Clover Dr S. W.
Cedar Rapids, IA
52404

FIRST CLASS MAIL
DATED MEETING NOTICE
SEP. 16 1990

UPPER RIGHT CORNER OF YOUR LABEL IS DATE OF LAST ISSUE.

LIST OFFICERS

PRES. HARVEY RAIT
TRES. ROBERT MALLOY
COR.SEC. JOHN PAZMINO
EDITOR. FRED STERN
LIBR. TOM SKAPINSKI

PLEASE SEND INQUIRIES TO:
LIST
MR. HARVEY RAIT
5 PERI LANE
VALLEY STREAM, N.Y. 11581
PLEASE SEND SUBMISSIONS TO:
LISTING
MR. FREDERIC STERN
214 ROBERTS ST.
HOLBROOK, N.Y. 11741

NYTSE

NYTSE MEETS THE MONDAY AFTER THE LIST MEETING AT;
MISS KIMS RESTAURANT
PARK AVENUE SOUTH
BETWEEN 21 ST. AND 22 ST.
MEETINGS START 7:30 PM.

COMING EVENTS:

SEPT. 9, 1990 LIST MEETING
SEPT.10, 1990 NYTSE MEETING

MEETING MINUTES
JUNE 10, 1990

HARVEY CALLED THE MEETING TO ORDER AT 2:30PM.

OUR NEXT MEETING WILL BE HELD AT HARVEYS HOUSE ON SEPT. 9.

SET OF 4 MICRODRIVE CARTRIDGES IN A CASE CAN BE PURCHASED AT LIST MEETINGS FOR $15.00.
IF YOU ARE INTERESTED SEE HARVEY

WE WERE GLAD TO SEE JAY SIEGEL AT THE MEETING AFTER A LONG ABSENCE. JAY ATTENDED THE MILWAUKEE EXPOSITION AND GAVE US A REPORT.
OF LOCAL INTEREST, ZEBRA SYSTEMS HAS MOVED FROM NEW YORK TO STATE CAMPUS PENNSYLVANIA.
JAY INFORMED US THAT HE WILL BE PUBLISHING A FDD NEWSLETTER.
IF YOU WANT TO SUBSCRIBE OR CONTRIBUTE, CONTACT JAY AT:
1274 49 ST.
APT. 821
BROOKLYN N.Y. 11219

WE RECEIVED A REPORT THAT HOME SHOPPING NETWORK WAS SELLING THE Z88.

NEWSLETTER UP-DATE

NUMEROUS ARTICLES FROM VARIOUS NEWSLETTERS WERE DISCUSSED.
THANKS TO TOM SKAPINSKI, PAGE 1 GIVES HIGH-LIGHTS OF THE DISCUSSIONS.

SWAP MEET

OUR SWAP-MEET ONCE AGAIN WAS A HUGH SUCCESS. JOHN PAZMINO WAS SELLING QL COMPUTERS, SOFTWARE AND PERIPHERALS. EDGAR GROSS AND FRED STERN HAD TS1000 STUFF.
TOM SKAPINSKI HAD A SALE ON BLANK PROGRAM TAPES (C-10), JUST TO MENTION SOME OF THE BARGAINS THAT COULD BE HAD.

OTHER BUSINESS

WORK IS PROCEEDING ON THE SECOND PRINTING OF ZX-81 AND TS1000 TECHNICAL TIDBITS. RELEASE DATE IS SEPT. 1, 1990.

CLASSIFIEDS

THIS CLASSIFIED SECTION IS AVAILABLE TO ALL LIST MEMBERS FREE OF CHARGE.
THE ONLY RESTRICTION IS THAT IT IS TO BE USED ONLY FOR THE SEEKING, SELLING OR SWAPPING OF SINCLAIR, TIMEX OR MICRORACE COMPUTER EQUIPMENT, PERIPHERALS AND SOFTWARE.
LISTING, LIST, AND ITS OFFICERS DO NOT ENDORSE, WARRANTY, OR GUARANTEE ANY OF THE ITEMS LISTED IN THIS CLASSIFIED SECTION

IF YOU HAVE A COPY OF Q-SAVE, FOR THE TS1000 PLEASE CONTACT FRED STERN 516-737-0963

NEEDS TS1000 RAM PACKS
D K STOICHEFF
605 MONTGOMERY
LAUREL MD 20707

I NEED ORIGINAL MANUAL TS2068
JESS WYDER
17 ACADEMY ST.
FISHKILL N.Y. 12524-1301

I AM LOOKING FOR A COPY OF (MASTERING MACHINE CODE ON YOUR ZX-81, BY TONI BAKER)
DONALD B LAMEN
RD 3 - BOX 3404
WINDSOR, N.Y. 13865

A FINAL WORD

MY NAME IS FRED STERN AND I AM THE EDITOR OF THIS EDITION OF LISTING.

I HOPE YOU ALL HAD A NICE SUMMER AS YOU KNOW, LISTINGIS NOT PUBLISHED DURING JULY AND AUGUST BUT WE ARE NOW BACK.

THANK YOUS GO TO TOM SKAPINSKI AND WARREN FRICKE FOR THERE CONTRIBUTIONS TO THIS ISSUE.

OUR ARTICLE BANK IS RUNNING LOW, PLEASE CONTRIBUTE NEWSWORTHY ARTICLES AND HELP SUPPORT YOUR SINCLAIR-TIMEX NEWSLETTER.

SEE YOU ALL AT THE NEXT MEETING.

JUNE NEWSLETTER ROUNDUP BY TOM SKAPINSKI

Corporate Systems Center, 730 North Pastoria Ave., Sunnyvale CA 94086

Selling: 720 K 3.5" Drives for $38.ea., $28.ea. in Quant. of 10

SNUG ?? What is hppening? I have read they published two newsletters. Our club should have joined by now.... perhaps we will hear more soon. However there is another Larken format public domain disk out. It is avaliable from Tony Willins at: P.O. Box 199, Vashon, WA 98070. (Three total at this time). Tape requests to Tim Ward,5142-D Ginkso Dr. SW., Tacoma, WA 98439 (7 60 min. tapes for TS-1000),(3 TS-2068 tapes) all items $4.00 ea. Also they are looking for contibutions to the Library.

PROFILE+5!, Profile update, New Features. Avaliable from RMG ENTERPRISES 1419 1/2 7th St., Oregon City, OR 97045 (Phone 503/655-7484)

Users Manual Available From Tom Woods $10.00 (ask RMG For Toms' address.

UPDATE Computer Systems, Quarterly Magazine for SINCLAIR computers, covers disk drives for our machines.... and is $18.00 P.O. Box 1095, Peru IN 46970.....Subscribe now for the upcoming year (starts next year with the issue due out after this current one) Confused ? ....write them

FAIRWARE 747 Flight Simulator For TS-2068 by Derek Ashton avaliable From Bob Swoger, 613 Parkside Circle, Streamwood IL 60107

COMMADORE 1351 Mouse may be used with TS-2068 if you have a LARKEN disk drive controller, use the Larken joystick port and hold the Right button while powering up the TS-2068, I also heard the CoCo mouse also works

Try it with Artworks and Art Studio programs, I believe the price For Commadore mouse is about $37.00...check prices before buying.

Advertisement! *Blank "New" C-10 Tapes,(no boxes) "IRISH" brand Twenty Four for $4.00 + $2.00 S/H. Thats right 24 for $4.00 + $2.00 S/H order from Tom Skapinski, 7 Atkinson Lane, Coram NY 11727 U.S.A. orders only.*

Advertisement! Blank C-10 tapes same as above pick up at meeting only SIX For $1.00 (no boxes, Index card only)

==========

Also LKDOS loader on LARKEN disk $5.00 PP From Tom SKapinki, This Menu type program will load your programs by just pressing ENTER when the highlighted bar is over your choice. No Keying in name and extensions. No typing errors! No File not Found errors! A very nice utility. You will be pleased if you try it!

SINCLAIR RULES! by John Pazmino

Ongoing news from the Mother Country about the Spectrum are good, good, good! There are now two nonSinclairs that have Spectrum emulation! The first is an allnew rig, from Miles Gordon Technology, that in one of its several opmodes turns into a Spectrum.

Many stateside Sinclair fans heard of this machine, the SAM Coupe', and it has been noted in some American Sinclair club newsletters. My own assessment, based on reports and correspondences from the UK, is that the Coupe' as a new computer system has little much to offer over existing systems. indeed, in British terms the Coupe' fully equiped costs as much as an Amiga or an Atari ST, both these being quite superior to the Coupe'. Never the less the inclusion of a Spectrum mode does speak well of the integrity and longevity of the Sinclair platform.

On the other hand, apparently the emulation is partial only. According to what I can discern the SAM Coupe' only allows Spectrum games to be played; I have not yet seen word of other normal Spectrum ops or use of Spectrum accessories on this instrument. Lending credence to this interpretation is the offering already of a Coupe' utility SAMTAPE that [supposedly] does render a complete emulation. It maps the Coupe' keyboard into a Spectrum's, with all the keywords and shifts. It allows running programs to be interrupted, POKEd and PEEKed, and SAVEd onto Coupe' discs, apparently like the Spectrum's own Multiface module.

Now comes the second instrument with Spectrum emulation: the Commodore Amiga!! A plugin module and Amiga disc constitute the MU kit. The module has a cassette port and the disc has the MU program. The module is only needed to feed Spectrum applications into the Amiga via the cassette socket. Once in the machine the program is saved direct to an Amiga disc whence it can be loaded and run thereafter. The drawback of this scheme is the slow workings, typical of SW MUs, like even Solution on the QL, and the B&W display.

There's more good gumbo about Sinclair! SW houses are looking again at the Spectrum, after saying two years ago that it is a declining system; the 16-bit systems were then the up & coming platform. But, but Elite, who dropped the Spectrum in 1987, are now fixing to offer Spectrum editions of their forthcoming 16-bit titles. Gremlin are massaging some of their existing 16-bit titles into Spectrum versions; first out of the mill is Supercars this summer. Palace are launching the Spectrum edition of its 16-bit 3D Pro Tennis.

Spectrum is dying? Not by the heavy sales rung up this past Christmas. About 30 kiloSpectrums were pushed thru the paypoints of the shops in England! Off-Island [meaning the British Isles] sales remain very strong. Spain, Holland, Italy all are strong Spectrum centers in Europe. There is a new market for Spectrum -- Eastern Europe. There was a trickle of Spectrums into the East all along, and into the USSR, too, but the economic scheme previously in force hindered and hampered sales. Now with the Wall down -- and the union of the German economy across that Wall -- Spectrums are snatched off the shelves in Prague, Budapest, Warsaw, and other East European centers.

A final bulletin. Your Sinclair, the UK's top-selling Sinclair mag, shifted from London to Bath in May 1990. This resulted from a leveraged buyout of the pub by Future Publishing for [variously reported] UKL 175 or UKL 180, about USD 300. Altho this seems low by American standards, do bear in mind the decrepit state of the British economy. What's more, this price did not allow for the costs of the move! The actual shifting was accomplished by applying the classical Swedish intrigue on BritRail. Anyway, Your Sinclair are domiciled at Future Publishing Ltd, 30 Mommouth Street, Bath BA1-2AP, England; tel 0255-442-244.

# #

The Logic Operator, NOT

For many of us, computer logic is a bit difficult to understand, and articles like the one reputed to be by Sharon Z. Aker in the July '88 issue of UPDATE muddies the water more. On page 14, under the title of Priorty, the article states that NOT B<C is interpreted as (NOT B)>C. This is wrong. NOT applies to the entire expression B<C as the < operator has a higher priority than NOT and the computer will evaluate B<C first. Look at page 228 of the manual for a priority listing.

NOT applied to a condition can result only in a logic value of 0 or 1, regardless of the appearance of the condition. For example, consider.....

```
NOT X = 50
```

This is interpreted by the computer as....

```
NOT (X = 50)
```

If X does equal 50, the condition, X = 50, is TRUE and results in a logic value of 1 for this condition. Then NOT 1 has a logic value of 0.

Conversely, if X has a value of anything other than 50, the condition is considered FALSE, and X = 50 results in a logic value of 0. Then NOT 0, in turn, has a logic value of 1, and the THEN action will take place.

We can show this by a short test program.....

```
10 LET X = 50
20 IF NOT X = 50 THEN PRINT
     "Yes"
30 IF NOT (X = 50) THEN
     PRINT "OK"
40 PRINT "End of test"
```

Now change the value of X in line 10 to anything but 50 and lines 20 and 30 will print out their strings. Why? Because X = 50 is not TRUE and its logic value becomes 0. Now, NOT 0 gives a logic value of 1. As the entire expression has a logic value of 1, the THEN action takes place.

The parentheses in line 30 are not necessary, but if you wish to make the evaluation of an expression clear, use them. The computer simply ignores superfluous parentheses, and evaluates X = 50 first because of the priorty of the = operator.

Let's go a step further. The computer considers all logic on a numerical basis. To the computer, every logic term that is TRUE is assigned the number 1; otherwise it gets a 0. We can put this to a test by adding lines such as the following to our test program.....

```
15 PRINT X = 50
25 PRINT NOT X = 50
35 PRINT NOT (X = 50)
```

and we will get nothing but the numbers 1 and/or 0, on printout of these lines.

You can get the same response by evaluating a logic statement yourself and substituting the resulting logical value in the computer line. Instead of line 20 as written above, write it as.....

```
20 IF 1 THEN PRINT "Yes"
```

and the computer will print the string. Change the 1 to 0 and the computer will not print the string. As we said before, the computer recognizes every thing but 0 as the number 1. So now use something like -3 in place of the 1. Again, the line will print out the string word.

If you get this concept of logic well established in your mind, you will have little or no trouble with logic NOT from here on in.

There is yet another way of treating logic NOT. You can change any expression wherein it appears to an equivalent one by recalling that it merely reverses logic TRUE and FALSE. In doing so, NOT drops out of the expression. So NOT X = 50 can be replaced by X <> 50. Shall we try another? OK. NOT Y > 10 can be replaced by Y <= 10. I personally don't prefer this way of handling NOT because we tend to forget how the computer itself treats this unique operator.

Warren Fricke

8 - 5 - 88

L.I.S.T.

L.I.S.T.

TIMEX LOGIC, PART I

This series of articles is called Timex Logic rather than computer logic because not all computers treat and/or use logic as fully as the Timex computer can. Some of the logic operations described in this series cannot be used on many other computers, including one fruity one, widely pushed onto our public school systems. This is not intended to imply that other computers cannot perform similar tasks. They can do the same thing by devious means but need more programming lines and consume more time in so doing. Appropriate logic can cut a lot of corners.

Line 10, that follows, is a statement containing a single logic condition. The construction of the line is typical of most, but not all, logic statements..... IF "some logic condition" THEN.....

```
10 IF X>5 THEN PRINT "OK"
```

X>5 is read X"greater than" 5, is a logic condition or relation.
X...is a variable that may have any numerical value.
> ..means "greater than" & is one of six symbol operators, =, <, >, <=, >=, and <>. Refer to page 228 of the manual for their meanings.

Line 10 means that if X is greater in value than 5, print the word "OK". Whatever follows THEN is the action to be taken, which can be almost anything. Simple as this statement seems to be, the computer must evaluate the logic condition of X>5 before it can act properly. It does this by making a comparison between X and 5, and then substituting a "logic value of 1" for the condition if the condition is TRUE, or a "logic value of 0" for the condition if it is FALSE. It follows that the condition will be TRUE or FALSE depending upon the value of the variable X at the time the computer works on this line, and the comparison is made.

Although the computer itself assigns a logic value of 1 to a TRUE condition, it considers any number other than 0 to be logical TRUE, even negative numbers. Thus a logic value of 0 denotes a FALSE condition and any other logical value denotes a TRUE condition. Hence, logic is number oriented. We can test this by adding a program line..

```
5  LET X=6
```

If we now RUN this two-line program, line 10 will print out the word "OK". This is because the logic condition of X>5 is TRUE, and the computer substituted a logic value of 1 for the condition. By direct entry have the computer PRINT X>5 and the response will be 1, the logic value of a TRUE condition.

If we change the value of X in line 5 to say 3 and RUN the program again, nothing prints out. This is because the computer assigned a logic value of 0 to the X>5 condition, as it is now FALSE. As before, and by direct entry, ask the computer to PRINT X>5. The response will be 0, the current logical value of X>5.

Now DELETE line 5. And in line 10 in place of the condition X>5, substitute 1, the logic value for a TRUE condition. RUN the one-line program and the word "OK" will print. It will also print "OK" no matter what numerical value is substituted for X>5, even negative values. Try some. But if a 0 is substituted for X>5, indicating a FALSE condition, nothing prints out.

It is important to realize that the conditions referred to as TRUE or FALSE do not involve morals nor are they indicative of desireability in any way. They are merely terms handed down from a branch of mathematics that was in existance long before the Timex computer. You can verify this by running thru the preceding exercises with the condition changed to read X<5, or X=5, etc.

NOT is unique as a logic operator. It is the only one of them that is also a function; so it always has a following argument (condition) and a subsequent result, which in this case is always 0 or 1. Unlike other functions its priorty is a low 4, ranking it below the six symbol operators and just above AND and OR.

NOT is used principally to inverse the logic value of another logic operation. In expressions like NOT X=Y, if X=Y is TRUE (logic value 1), NOT X=Y is FALSE (logic value 0). If X=Y is FALSE, then NOT X=Y is TRUE. In this combination, the Timex computer evaluates the operation X=Y first, as the logic symbol = has a priorty higher than NOT. Parenthes (high priorty) around the operation X=Y are not needed to do this, but may be used sometimes for clarity. The Timex computer simply ignores superfluous parentheses.

NOT can be eliminated, if desired, by inversing the expression that it modifies. For example, NOT X=Y can be replaced by X<>Y. But use care in so doing. The inverse of X>Y is X<=Y, and not simply X<Y.

There are times when NOT can not conveniently be eliminated. For example, NOT X means that if X has a value of 0, NOT X has a logic value of 1. If X has a value other than 0, NOT X has a logic value of 0. From this one can appreciate that NOT "something" will always have a logic value of 0 or 1, and nothing else.

In Part II we will show how compound logic statements can be built up using the combining logic operators, AND and OR. And, how the Timex computer considers the priorty of all operations, including those assigned to logic conditions, so that it can do "first things first"; thus arriving at the proper action to take in any situation.

Warren Fricke
8-9-88

TRANSLATING THE DATA STATEMENT

A commonly found instruction in most versions of BASIC is DATA and its cohort READ and RESTORE. For some reason Sinclair BASIC doesn't include them. This can cause problems for the user trying to translate a program from another mach to the T/S.

The Boston Computer Society's Sinclair Timex Newsletter published the follow translation routine and we reprint it with their kind permission.

The DATA statement allows a programmer to define a list of numbers in his program. The READ statement retrieves these numbers one at a time, and assigns them to a variable. The RESTORE statement forces the READ to scan list from the beginning again.

In order to translate a program which has DATA statements, you must:

1. Include the following routine in your program:

```
500   LET DI=DI+1
510   IF D$(DI)=" "THEN GOTO 500
520   LET DJ=DI
530   LET DJ=DJ+1
535   IF DJ> LEN D$ THEN GOTO 550
540   IF D$(DJ) <> " " THEN GOTO 530
550   LET D=VAL D$(DI TO DJ-1)
560   LET DI=DJ
570   RETURN
```

2. Have a statement LET DI=0 somewhere near the beginning of your program (before doing any READs). Follow this by the DATA statement LET D$="your data here". The data elements (i.e. numbers) are separated by blanks. For example, if the program being translated has DATA 5, 10.4, -3, 6.9E-10, then use LET D$="5 10.4 -3 6.9E-10.

3. Replace RESTORE statements in the program by LET DI=0.

4. Finally, replace READ statements by GOSUB 500 followed by LET variable=D. For example, READ A becomes GOSUB 500 and LET A=D. Error 3 results if you read past the end of the data.

I am writing to ask you what the error report code H means. For after typing in a program on my ZX81 which included machine code this was displayed and it quite baffled me for it is not detailed in the ZX81 manual. Please could you enlighten me.

Darren Crook
Velindre
Wales

Darren
This is what's known as a crash. Usually with the average ZX81, crash looks much more hi-res-and-modern-art-mode, but this crash has somehow called the error report routine, and the report H has been thrown out because the report subroutine has been called in a non-standard way. What you must do is check the program listing with the machine code you already have in your computer. This is probably where the problem lies.

I would be pleased if you could print a copy of the "Conversion Table", for converting ZX81 PEEK values to that of the Spectrum.

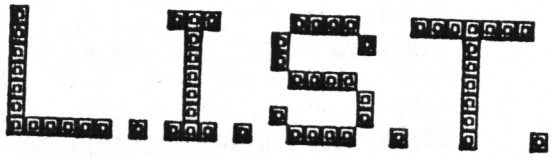
In various books I have seen such tables to convert ZX80 PEEK values to the ZX81, but I have not yet seen one for the ZX81 to Spectrum.

Although I have both the Spectrum and ZX81, I much prefer the ZX81 with 16K memory, as the programs devised for it are both short and sweet, not as long-winded and bug-ridden as Spectrum programs. Thus, the Spectrum lies neglected until I convert PEEK values from better programs for it.

J T Clench
Hall Green
Birmingham

Mr Clench
I took your letter as a challenge and sat down and worked out the necessary conversions. Here they are for the first time, as you and I believe, in print at last!

ZX81	SPECTRUM
16384	23610
16385	23611
16386	23613
16388	23730
16390	23617
16391	23617
16393	NO EQUIVALENT
16394	23621
16396	NO EQUIVALENT
16398	23684
16400	23627
16402	23629
16404	23641
16406	23645
16408	23647
16410	23651
16412	23653
16414	23655
16415	23656
16418	23659
16419	23660
16421	23560
16424	NO EQUIVALENT
16425	23637
16427	23662
16429	23665
16430	23666
16432	23668
16434	23670
16436	23672
16438	23677
16439	23678
16440	23680
16441	23688
16442	23689
16443	NO EQUIVALENT
16444	23296
16477	23698



LOADING/SAVING PROGRAMS
8K ROM

SAVING. Keep the tape-computer connector always plugged into the EAR and MIC receptacles of the computer. However at the tape recorder end only keep the one connected which you are using. Let us say the last line of your program is 1000 SAVE "SHOOTIST" . Then press LIST 1000 and ENTER. Then hit SHIFT 1 (for EDIT). When line 1000 appears at bottom screen DELETE (SHIFT Ø) the 1000. Move the tape to the position you want it. Then voice record the word SHOOTIST three times and immediately stop the tape. Set the tape volume to level which saves best. Start recording and immediately press ENTER. The computer can be close to the TV but the tape recorder must be maximally (connector at full stretch) remote from the TV and computer. Only the MIC side of the connector is plugged into the tape recorder. You can have the TV sound up a bit as you record. If you have a long 16K program saving you can do other things and when the buzzing sound stops and there is a 0/0 report bottom screen stop the tape and you have saved a program. Note the start and stop tape position and mark them in a catalog together with the SAVE title. Replay some of the tape to make sure you hear the shrill buzz of program (MIC plug pulled out).

LOADING. LOAD in FAST mode. Type in the command LOAD "SHOOTIST" . Locate your program on the tape from your catalog. After you hear the title (it should be three times) there is a faint buzz. In a few seconds there is total silence. Stop the tape at the beginning of the silence. Plug into the tape the EAR side of the connector. Leave the MIC plug not connected to the tape. Adjust tape volume to the known proper level. Push the tape PLAY switch and immediately press ENTER on the ZX. Have the TV sound up a bit and when the buzz stops and there is the 0/0 report bottom screen stop the tape and you have loaded the computer.

SAVING FAILS. Check to see the tape volume is high enough. Check to see that you are using the MIC connector. Is the MIC plug in the right receptacle? Is the tape remote from the TV and ZX? Reset the voice title and SAVE again...it should work after all these checks.

LOADING FAILS. Do the above checks and also put the computer end of the connectors in and out a couple of times to shine the contacts. If you have a 16K RAM pack put it on and off twice to shine contacts. Sometimes the tape is stretched or glitched in other ways. Programs should always be saved twice. Do not load or save on metal tables.

CONNECTOR. The ZX comes with an 18 inch tape-ZX connector. The older ones had screw in tips which break after much use. The later deliveries are with unitary plastic tips and are more durable. In persisent failures check connector for breaks (this happened to one of our older models).

SAVING SCREEN DISPLAYS.

```
  10 PRINT "THIS DISPLAY WILL ER
ASE WHEN YOU HIT SAVE ""TEST"""
  20 SAVE "TEST"
```

```
  10 INPUT A$
  20 PRINT "THIS DISPLAY WILL NO
T ERASE WHEN YOU HIT SAVE"
  30 IF INKEY$="S" THEN SAVE A$
  40 GOTO 30
```

The left program is the regular way of saving. Try it and see that the screen display is lost when saving is finished. RUN the right program. INPUT any string (any set of one or more characters) and hit ENTER. Press SAVE and notice after the saving the display remains on screen. IT will be saved. Remember you need the 30-40 continuing loop to make line 30 work from the keyboard. The double quotation marks are SHIFT Q.

FREQUENT SAVES. When composing or entering a long program SAVE after every 50 or 60 lines to avoid unfortunate losses from wipouts (crashes).

PAUSE CAVEAT. With ZX80/8K ROM or ZX81 FAST mode right after a PAUSE line you need a POKE 16437,255 line or crash happens in a few minutes.

VARIABLES AND ARRAYS. In a long program with large arrays and many variables (all these arise from LET statements) saving is prolonged because these items too are saved. Press CLEAR and ENTER before you save and all these will be cleared away shortening saving time. A shorter program has less chance to be glitched. If you want to save these items because they would be hard to regain or type in then do this: For the second last line of the program insert a REM statement (the last line is the SAVE line). Before you save hit LIST (REM line number) and ENTER. Then SAVE program. Later when loading this program hit ENTER and the last two lines will appear on screen. The REM line should be something like 990 GOTO 100 TO PRSRV VARBLS . RUNning a program wipes out all variables. You use a GOTO statement to preserve them and the GOTO number must be past all the DIM and LET X=Ø type statements or there will be clearances. The REM line reminds you not to RUN when you first load the program.

LARGE TO SMALL. You cannot LOAD a small program SAVED from a 16K ZX into a 1K ZX. You can from 1K to 16K. Large to small no,small to large yes.

COMMONEST CRASHES: jiggling transformer jack or RAM Pack.
SPEED TIP: In loading type LOAD "" and next program on tape will LOAD.

BASIC AT A STRETCH

Tired of using the same old commands? Well, saddle up your Interface 1 unit and let Gavin Smyth demonstrate the practice of adding up to 26 new commands to your Spectrum.

Sinclair Basic, for all its slowness, is fairly comprehensive. But as with all things, the more you have, the more you want and a few extra commands might certainly come in useful. How about, for instance, an absolute draw. With Interface 1 attached, it's possible to add as many extra commands as you like.

First, let's look at the extended interpreter and the individual routines in detail.

NEW INTERPRETATIONS

When a line of Basic is entered, the original ROM checks its syntax; if it fails to recognise the command, it passes control over to the Interface 1 ROM. This scans the line checking for new words such as 'FORMAT'. If this test also fails, the processor jumps to the error handling routine (at ERR6) via the vectored address at 23735. You can alter this address and have further tests implemented to provide extra commands.

This is what the extended interpreter does. It checks for the presence of an asterisk at the beginning of a statement followed by a letter; with this simple test, however, only 26 new commands can be invented. If it finds no asterisk or letter it gives the usual syntax error. If all is correct, it jumps to the actual routine via the table of vectors. Note that unimplemented commands point to the error handler. The individual command routines check the rest of the syntax and contain the runtime routine.

For the interpreter to function properly, there must be at least one space between the new command word and any following arguments. The command word must start with an asterisk and a letter, followed by any (or no) characters at all.

ABSOLUTE DRAW

The first new command is:

***DRAW x,y**

Which draws to the point (x,y). It's much simpler to use than relative co-ordinates (especially in graphs). Thus:

PLOT 100,100: DRAW 20,30

Is equivalent to:

PLOT 100,100: *DRAW 120,130

First, the routine calls SPACE to get to the end of the first word (in this case, *DRAW). Then it checks for the presence of two numeric arguments separated by a comma; if there's something wrong it jumps to the error handler. In syntax time — that is, when the line is first entered into a program — the routine ends here; in runtime the rest is executed. This part simply calculates the size of the relative co-ordinates and calls the Basic ROM's DRAW subroutine.

When this subroutine is being executed, the Interface 1 ROM is paged in. This allows routines in the Interface 1's ROM, such as STEND, to be simply CALLed; routines in the main ROM, such as FNDINT1, have to be called via

```
    1 REM ** EXTENDED BASIC **
    2
    3 REM In line 20, set start
to the desired beginning of the
machine code, and in line 10,
CLEAR to the location before
    4
    5 REM This BASIC program will
relocate the code anywhere in
memory, but it is fairly slow
    6
    7 REM The machine code is 559
bytes long
    8
   10 CLEAR 64797
   20 LET start=64798
   30 LET a$=""
   40 FOR a=start TO start+558
   50 IF a$="" THEN  READ a$
   60 POKE a,FN h(a$)*16+FN h(a$(
2))
   70 LET a$=a$(3 TO )
   80 NEXT a
   99 REM Now relocate the code
  100 LET a=20: LET p=45: GO SUB
500
  110 LET a=53: LET p=99: GO SUB
500
  120 LET a=77: LET p=165: GO SUB
500
  130 LET a=83: LET p=267: GO SUB
500
  140 LET a=97: LET p=465: GO SUB
500
  150 LET a=100: LET p=31: GO SUB
500
  160 LET a=166: LET p=31: GO SUB
500
  170 LET a=268: LET p=31: GO SUB
500
  180 LET a=463: LET p=394: GO SU
B 500
  190 LET a=466: LET p=31: GO SUB
500
  200 PRINT "To use the extra com
mands, enter"
  210 PRINT '"POKE 23735,";start-
256*INT (start/256);": POKE 2373
6,";INT (start/256)
  220 STOP
  499 REM This routine sorts out
the absolute addresses to
relocate the program
  500 LET address=start+a+1
  510 LET pointsto=start+p
  520 POKE address,INT (pointsto/
256)
  530 POKE address-1,pointsto-256
*PEEK address
  540 RETURN
  899 REM function to convert a
hex digit to its decimal value
  900 DEF FN h(a$)=CODE a$-48-7*(
a$>="A")
  999 REM data for the machine
code program

 1000 DATA "D71800FE2AC2F001D720"
 1010 DATA "00E69FFE1B3801AF8721"
 1020 DATA "2DDB06004F095E2356EB"
 1030 DATA "E9D77400FE20C8FE3AC8"
 1040 DATA "FE0D20F3C9F001F001F0"
 1050 DATA "01F00163D8F001F001F0"
 1060 DATA "01F001F001F001F001F0"
 1070 DATA "01F001F001F001A5D8F0"
 1080 DATA "01F0010BD9F001F001F0"
 1090 DATA "01F001F001F001D1D9CD"
 1100 DATA "1FD8D7821CFE2CC2F001"
 1110 DATA "D72000D7821CCDB705D7"
 1120 DATA "941E217E5C9638141601"
 1130 DATA "47C5D5D7941ED1C1217D"
 1140 DATA "5C96380A1E01180A16FF"
 1150 DATA "ED4418EB1EFFED444FD7"
 1160 DATA "BA24C3C105CD1FD8D782"
 1170 DATA "1CFE2CC2F001D72000D7"
 1180 DATA "821CFE2CC2F001D72000"
 1190 DATA "D7821CCDB705D7941ECB"
 1200 DATA "27CB27CB2716005F2A7B"
 1210 DATA "5C19E5D7941EF5D7941E"
 1220 DATA "C14FD13E08F5C5D7AA22"
 1230 DATA "F5E5D5D74D0DD7DB0BE1"
 1240 DATA "46D1EBF10E003CCB38CB"
 1250 DATA "193D20F9702371C10513"
 1260 DATA "F13D20D9C3C105CD1FD8"
 1270 DATA "D7821CCDB705D7941EE6"
 1280 DATA "032B0AFE012B2CFE0228"
 1290 DATA "1518C3EC0210040A706"
 1300 DATA "20CB1E2310FB3D20F5C3"
 1310 DATA "C1053EC021FF57A70620"
 1320 DATA "CB162B10FB3D20F5C3C1"
 1330 DATA "05A71100400603C53E08"
 1340 DATA "083E07626B24E5012000"
 1350 DATA "EDB0D13D20F3010007ED"
 1360 DATA "42E5012000EDB0D1083D"
 1370 DATA "20E001E00609545D0120"
 1380 DATA "00ED42EBEDB0C110CC21"
 1390 DATA "E057062077231DFCC3C1"
 1400 DATA "0511FF570603C53E0808"
 1410 DATA "3E07626B25E5012000ED"
 1420 DATA "B8D13D20F301000709E5"
 1430 DATA "012000EDB8D1083D20E1"
 1440 DATA "01E006ED42545D012000"
 1450 DATA "09EBEDB8C110CD210040"
 1460 DATA "0620C38AD9CD1FD8D782"
 1470 DATA "1CCDB705D7941EA72832"
 1480 DATA "FE012809FE022B0DFD36"
 1490 DATA "000AEF01B001211000 8"
 1500 DATA "0601FFFE21201811040 0"
 1510 DATA "3E10C5D5E5F5D7B5C3F1"
 1520 DATA "E1D1C1093D20F1C3C105"
 1530 DATA "F33A485C0F0F0F260446"
 1540 DATA "2B1DFED3FEEE10087CB5"
 1550 DATA "28030818F0FBC3C105"
```

BASIC ATA STRETCH

This is the machine code disassembly of a series of extended Basic routines which expand Sinclair Basic by up to 26 new commands, of the form '*name', ie., commencing with an asterisk, followed by a letter (upper or lower case) or word and any necessary parameters. (Note that this program only works with Interface 1 attached, and the system variable vectors at 23735 and 23736 should be set to point to START.)

```
START RST 10 GETCHAR
      CP "*"
      ;make sure 1st char is *
      JP NZ ERR6
      RST 10 NXTCHAR
      AND 9F
      CP +27
      ;check 2nd is within
      ;alphabetic range
      ;(very simple test so may
      ;fail with some chars)
      JR C INDEX
      XOR A
INDEX ADD A,A
      LD HL,TABLE
      LD B,0
      LD C,A
      ADD HL,BC
      LD E,(HL)
      INC HL
      LD D,(HL)
      EX DE,HL
      JP (HL)
      ;jump to specific routine
      ;for each of the new 26
      ;commands
      ;subroutine to find the
      ;end of a word
SPACE RST 10 CHADD
      CP " "
      RET Z
      CP ":"
      RET Z
      CP 0D
      JR NZ SPACE
      RET
```

The table of vectors to the specific routines.

```
TABLE ERR6   ;error vector
      ERR6   ;vector for *A...
      ERR6   ;*B...
      ERR6   ;*C
      DRAW   ;*D
      ERR6   ;*E
      ERR6   ;*F
      ERR6   ;*G
      ERR6   ;*H
      ERR6   ;*I
      ERR6   ;*J
      ERR6   ;*K
      ERR6   ;*L
      ERR6   ;*M
      ERR6   ;*N
      ERR6   ;*O
      PRINT  ;*P
      ERR6   ;*Q
      ERR6   ;*R
      SCROL  ;*S
      ERR6   ;*T
      ERR6   ;*U
      ERR6   ;*V
      ERR6   ;*W
      ERR6   ;*X
      ERR6   ;*Y
      ZAP    ;*Z
```

This routine provides the command to carry out absolute drawing. It is used in the form *DRAW x,y (or *d x,y), where x and y are the co-ordinates of the point to be drawn to.

```
DRAW  CALL SPACE
      RST 10 EXPT1NM
      CP ","
      ;check for separator
      JP NZ ERR6
      RST 10 NXTCHAR
      RST 10 EXPT1NM
      CALL STEND
      ;now x & y are on the top
      ;of the stack
      RST 10 FNDINT1
      ;y co-ord into A
      LD HL,COORDS+1
      ;address of previous y
      SUB (HL)
      ;find relative y
      JR C DRAW2
      ;if -ve, adjust it
      LD D,1
DRAW1 LD B,A
      PUSH BC
      PUSH DE
      RST 10 FNDINT1
      POP DE
      POP BC
      LD HL,COORDS
      SUB (HL)
      ;same for x co-ord
      JR C DRAW3
      LD E,1
      JR DRAW4
DRAW2 LD D,FF
      NEG
      JR DRAW1
DRAW3 LD E,FF
      NEG
DRAW4 LD C,A
      RST 10 DRAWR
      ;perform actual drawing
      JP END1
```

You can print a UDG anywhere on the screen using the routine below. You access this ability using the command *PRINT x,y,c (or *p x,y,c), where (x,y) is the pixel co-ordinate of the top left of the character and c is the number of the UDG (A is '0', B is '1', and so on).

```
PRINT CALL SPACE
      RST 10 EXPT1NM
      CP ","
      JP NZ ERR6
      RST 10 NXTCHAR
      RST 10 EXPT1NM
      CP ","
      JP NZ ERR6
      RST 10 NXTCHAR
      RST 10 EXPT1NM
      CALL STEND
      RST 10 FNDINT1
      SLA A
      SLA A
      SLA A
      LD D,0
      LD E,A
      LD HL,(UDG)
      ADD HL,DE
      ;HL contains the start
      ;of the required UDG data
      PUSH HL
      RST 10 FNDINT1
      PUSH AF
      RST 10 FNDINT1
      POP BC
      LD C,A
      ;BC contains the co-ords
      ;of the point to print to
      POP DE
      LD A,8
PRNT1 PUSH AF
      PUSH BC
      RST 10 PIXADD
      ;convert co-ords into an
      ;address and pointer in A
      PUSH AF
      PUSH HL
      PUSH DE
      RST 10 TEMPS
      ;set up colours
      RST 10 POATTR
      ;fill in attributes
      POP HL
      LD B,(HL)
      POP DE
      EX DE,HL
      POP AF
      LD C,0
      ;BC contains a byte of
      ;UDG data
      INC A
PRNT2 SRL B
      RR C
      DEC A
      JR NZ PRNT2
      ;shift data along to the
      ;correct pixel within the
      ;screen byte
      LD (HL),B
      INC HL
      LD (HL),C
      ;put it on the screen
      POP BC
      DEC B
      INC DE
      ;next line
      POP AF
      DEC A
      JR NZ PRNT1
      JP END1
```

This part of the program provides a pixel scroll in any of four directions; the command is used in the form *SCROLL n (or *s n), where n specifies direction.

```
SCROL CALL SPACE
      RST 10 EXPT1NM
      CALL STEND
      RST 10 FNDINT1
      AND 3
      ;take n mod 4
```

```
        JR Z SCRL0
        ;scroll right
        CP 1
        JR Z SCRL1
        ;scroll up
        CP 2
        JR Z SCRL2
        ;scroll left
        JR SCRL8
        ;scroll down
SCRL0   LD A,+192
        ;no of lines on screen
        LD HL,+16384
        ;start of screen
SCRL3   AND A
        ;clear carry
        LD B,+32
        ;no of bytes per line
SCRL4   RR (HL)
        ;shift 1 bit right
        INC HL
        ;next byte on line
        DJNZ SCRL4
        DEC A
        ;next line
        JR NZ SCRL3
        JP END1
SCRL2   LD A,+192
        LD HL,+22527
        ;end of screen
SCRL5   AND A
        LD B,+32
SCRL6   RL (HL)
        ;shift 1 pixel left
        DEC HL
        DJNZ SCRL6
        DEC A
        JR NZ SCRL5
        JP END1
SCRL1   AND A
        ;reset carry
        LD DE,+16384
        ;1st byte of screen
        LD B,3
        ;no of 'sections' to
        ;be scrolled
SCRL7   PUSH BC
        LD A,8
        ;no of character lines
        ;per section
SCRL8   EX AF
        LD A,7
        ;no of pixel lines per
        ;character - 1
SCRL9   LD H,D
        LD L,E
        INC H
        ;HL points to next pixel
        ;line
        PUSH HL
        LD BC,+32
        LDIR
        POP DE
        DEC A
        JR NZ SCRL9
        LD BC,+1792
        SBC HL,BC
        ;adjust pointer to start
        ;of next character line
        PUSH HL
        LD BC,+32
        LDIR
        POP DE
        EX AF
        DEC A
        JR NZ SCRL8
        LD BC,+1760
        ADD HL,BC
        LD D,H
        LD E,L
        LD BC,+32
        SBC HL,BC
        ;adjust pointer for next
        ;screen section
        EX DE,HL
        LDIR
        POP BC
        DJNZ SCRL7
        LD HL,+22496
        LD B,+32
        ;now clear the last line
SCRLA   LD (HL),A
        ;A already contains 0
        INC HL
        DJNZ SCRLA
        JP END1
SCRLB   LD DE,+22527
        ;last location on screen
        LD B,3
SCRLC   PUSH BC
        LD A,8
SCRLD   EX AF
        LD A,7
SCRLE   LD H,D
        LD L,E
        DEC H
        PUSH HL
        LD BC,+32
        LDDR
        POP DE
        DEC A
        JR NZ SCRLE
        LD BC,+1792
        ADD HL,BC
        PUSH HL
        LD BC,+32
        LDDR
        POP DE
        EX AF
        DEC A
        JR NZ SCRLD
        LD BC,+1760
        SBC HL,BC
        LD D,H
        LD E,L
        LD BC,+32
        ADD HL,BC
        EX DE,HL
        LDDR
        POP BC
        DJNZ SCRLC
        LD HL,+16384
        LD B,+32
        JP SCRLA
        ;clear top line
```

This last routine provides better sound effects. The command is used in the form *ZAP n (or *z n), where n specifies whether you require an explosion, a falling tone or a rising tone.

```
ZAP     CALL SPACE
        RST 10 EXPT1NM
        CALL STEND
        RST 10 FNDINT1
        AND A
        JR Z ZAP0
        CP 1
        JR Z ZAP1
        CP 2
        JR Z ZAP2
        LD (IY+0),0A
        RST 28
        ;integer out of range
        ;error code
ZAP1    LD BC,180
        LD HL,10
        JR ZAP3
ZAP2    LD BC,FEFF
        LD HL,1820
ZAP3    LD DE,4
        LD A,10
ZAP4    PUSH BC
        PUSH DE
        PUSH HL
        PUSH AF
        RST 10 BEEPER
        ;make short burst of tone
        POP AF
        POP HL
        POP DE
        POP BC
        ADD HL,BC
        ;change frequency
        DEC A
        JR NZ ZAP4
        JP END1
ZAP0    DI
        LD A,(BORDCR)
        RRCA
        RRCA
        RRCA
        ;A=border colour
        LD H,4
ZAP5    LD B,(HL)
        DEC HL
ZAP6    DJNZ ZAP6
        ;'random' delay to
        ;produce white noise
        OUT FE,A
        ;activate speaker
        XOR 10
        ;toggle speaker on/off
        EX AF
        LD A,H
        OR L
        JR Z ZAP7
        EX AF
        JR ZAP5
ZAP7    EI
        JP END1
```

RST 10h followed by the starting address (this is true for all the new command routines).

PRINTING PIXELS

The next routine allows a user-defined graphic character to be placed anywhere on-screen. The reason for restricting it to UDGs is that it's unlikely that anyone would want to print a lot of text like this (since the routine handles only one character at a time) and, if required, the UDGs may be re-defined as letters.

The syntax for the command is:

```
*PRINT x,y,c
```

Where (x,y) are the pixel co-ordinates of the top left corner of the character (x lies between zero and 255, y between zero and 175) and c is the number of the UDG (UDG A is zero, UDG B is one . . . UDG U is 20). For example:

```
FOR x=0 TO 247: *PRINT x,100,0: NEXT x
```

The above will glide the first UDG (initially a capital 'A') across the screen. This time the extended syntax checker looks for three numbers separated by commas.

In the runtime routine, the data for the UDG is found and shifted across to give the correct information for the screen memory; the listing contains the code and full comments. The routine ends with the attribute bytes being set to the permanent colours.

PIXEL SCROLL

The scroll command allows pixel scrolling of the entire screen. The actual command is:

```
*SCROLL n
```

Where n is a numeric expression controlling the direction of the scrolling (n must lie between zero and 255, but only the mod four value is used):

n=0 (or 4,8,. . .) scrolls right one pixel
n=1 (or 5,9, . . .) scrolls up
n=2 scrolls left
n=3 scrolls down

For example, to scroll one whole character down, you could use:

```
FOR f=1 TO 8: *SCROLL 3:NEXT f
```

This moves the screen by eight pixel lines. The syntax checker this time looks for only one numeric argument. Only the lower two bits are used (giving a range of zero to three) and the routine jumps to each separate scrolling routine. The left and right scrolls are fairly simple — they just shift each line of 32 bytes by one bit. The up and down scrolls are complicated by the layout of the screen memory, but all they do is move each line into the one above or below, clearing the final one.

The colour attributes are not affected at all, since they have only character block and not pixel resolution.

SIMPLE SOUNDS

The final routine is a simple sound effects generator — useful for games if nothing else. The command syntax is:

ROUTINE	VALUE	COMMENT
GETCHAR	18	Fetches current char on BASIC line into A
NXTCHAR	20	Fetches next char, ignoring spaces
ERR6	1F0	Normal error handler
STEND	5B7	Subroutine to exit command routine in syntax check time
END1	5C1	Routine to exit in runtime
EXPT1NM	1C82	Syntax checker for expecting a numeric expression (puts it on the stack)
FNDINT1	1E94	Fetches number into A from stack
BEEPER	3B5	Loudspeaker routine HL=437500/freq-30.125 DE=freq*duration
CHADD	74	Reads next character including spaces
BORDCR	+23624	System variable holding attributes for the lower screen and border
STACKA	2D28	Puts A on to the calculator stack
COORDS	+23677	System variable holding x & y co-ordinates of last point plotted
DRAWR	24BA	Relative draw routine, takes x & y offsets from BC & DE registers
PIXADD	22AA	Converts co-ords in BC to an address in HL
UDG	+23675	System variable holding address of user-defined graphics data
POATTR	BDB	Sets attributes for the screen byte in HL
TEMPS	D4D	Sets temporary colours (used by POATTR) to the permanent colours

The ROM routines and addresses used by the program.

```
*ZAP n
```

Where n, between zero and two, specifies the type of sound:

n=0 gives an explosion noise
n=1 gives a falling tone
n=3 gives a rising tone

For example:

```
FOR f=1 TO 100: *ZAP INT(RND*3):NEXT f
```

The above will give several seconds of 'exciting' sounds (well, more exciting than BEEP!). The computer again looks for a single argument and checks its value; if it's out of range, it gives an error, otherwise it jumps to the specific routine. The rising and falling tones are produced by short beeps of changing frequencies, while the explosion is produced by sending 'random' data from the ROM to the speaker port.

A COMMANDING LEAD

You can check from the assembler listing several important points about adding new commands:

1. First, the syntax is checked — using EXPT1NM for numeric expressions and CP "," to check for separating commas. This section ends with CALL STEND.
2. Next comes the runtime routine and, since the values of the arguments are on the calculator stack, these can be pulled off into the A register with FNDINT1.
3. The runtime code ends with a jump to END1.
4. Any subroutine in the Basic ROM must be called via RST 10h. Note that all registers are preserved while the ROMs are paged.
5. Before the code will run, the vector at 23735 must point to the beginning, with a jump back to ERR6 at the end to trap genuine syntax errors.

The code may be entered using either an assembler or the Basic program provided which will relocate the 559 bytes of machine code anywhere in memory; it should run on a 16K Spectrum, so long as Interface 1 is connected.

To use it, set lines 10 and 20 to the desired starting address (for example, 31000 in a 16K machine and 64700 in a 48K model). Next SAVE the program in case there is a mistake (which is usually fatal in machine code). Now, RUN the program and after a few minutes a message will appear on the screen; the code has been located in memory but will have no effect since the vector at 23735 has not been altered. To use the extra commands enter the line in the message — as a single line rather than two separate POKEs, otherwise the machine may crash.

If NEW is entered the vector is reset; however, the code is still in memory, so repeat the two POKEs to allow the commands to be used. We've seen a simple interpreter that will allow up to 26 new commands, but with only four examples; there are 22 more possible words, so get working!

ASSEMBLER NOTES

The assembler used was the Artic version which, in fact, is slightly non-standard. The main differences between it and a standard assembler are:

EQU is replaced by the '=' sign.
All numbers are in hexadecimal unless preceeded by '+' or '−', in which case they are in decimal,
DEFB, DEFW, and so on do not exist; the number or label is simply placed in the source text where the DEFB would occur, and EX AF,AF is entered as EX AF. ▼s